**Disclaimer:** this report is not the full version and shortened to show my skills and learning…

1. **Introduction**

This report documents the design, implementation, and testing of a Formula 1 racing database. It includes key entities such as championships, races, teams, drivers, and results. The goal was to create a well-structured relational database that supports complex queries for performance analysis, standings tracking, and event management. I built the database step by step, made changes all around.

This report covers an overall comprehensive the process:

* **Database design** (ER modelling, relational schema, table descriptions, normalization)
* **Implementation** (tables, constraints, sample data)
* **Query design** (basic and advanced queries for analysis)
* **Advanced features** (procedures, triggers, views)
* **Python integration** (connecting and executing queries programmatically)
* **Reflections** on challenges and lessons learned

When first brainstorming for this project, I took the time to understand how Formula 1 works, including the structure of teams, races, and championships, as I was not aware of it to begin with. This definitely helped me identify the key entities and relationships required for my database. I then examined relevant data sources provided and looked for things beyond and wrote down a schema while considering necessary constraints, like foreign keys. Having a clear design from the beginning helped streamline the implementation process.

1. **Database Design**
2. **ER Modelling**

**Firstly, entity sets, their keys and attributes:**

|  |  |  |
| --- | --- | --- |
| **Entity Set** | **Key** | **Other Attributes** |
| Championship | Championship\_ID (PK) | Year |
| Grand Prix | Race\_ID (PK) | GP\_Name, Date, Location (Country, City, Circuit\_Name), Distance |
| Team | Team\_ID (PK) | Name, Country\_T, Principal |
| Driver | Driver\_ID (PK) | Name, Nationality, DOB |
| Ticket | *(None)* | Average\_Price\_USD, |
| Car | Car\_ID (PK) | Power\_Unit, Chassis, Model\_Name |

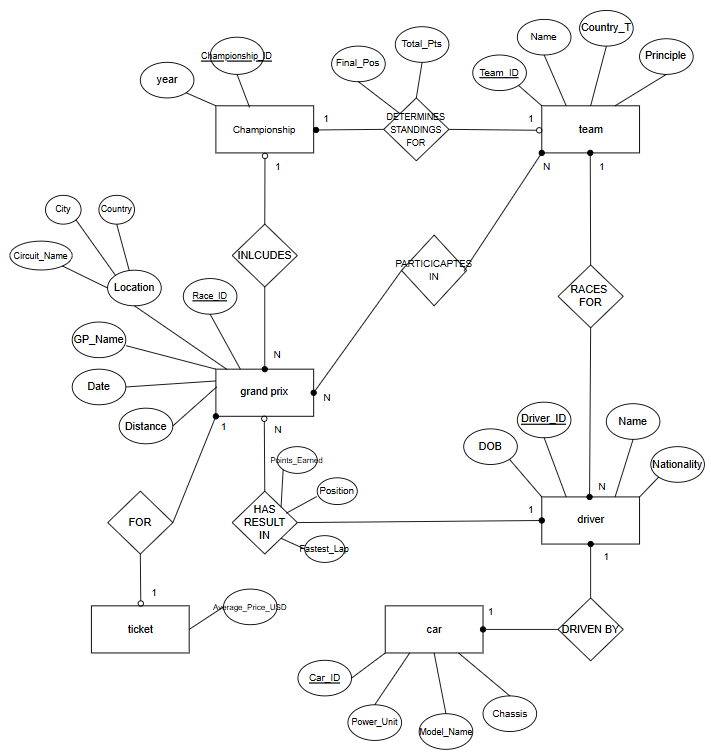
**Relationship sets, their participating entity sets and attributes:**

|  |  |  |
| --- | --- | --- |
| **Relationship Set** | **Between Which Entity Sets** | **Attributes of Relationship Set (if any)** |
| Includes | Championship, Grand Prix | None |
| Races for | Driver, Team | None |
| Participates in | Team, Grand Prix | None |
| Has a result in | Driver, Grand Prix | Position, Points\_Earned, Fastest\_Lap |
| for | Ticket, Grand Prix | None |
| Determines Standings for | Team, Championship | Final\_Position, Total\_Points |
| Driven by | Car, Driver | None |

**Cardinality and participation constraints**

|  |  |  |
| --- | --- | --- |
| **Relationship Set** | **Cardinality Constraints** | **Participation/Other Constraints** |
| Includes (Championship, Grand Prix) | One-to-Many (One Championship includes many Grand Prix, each Grand Prix belongs to one Championship) | Mandatory for Grand Prix (every race is part of a championship), Optional for Championship (a championship must have races, but have it it without specific races at first) |
| Races for (Driver, Team) | Many-to-One (Many Drivers race for one Team, but a Driver can only be in one Team at a time) | Mandatory for both (each Driver must belong to a Team, and a Team must have at least one Driver). |
| Participates in (Team, Grand Prix) | Many-to-Many (A Team participates in many Grand Prix, and each Grand Prix has multiple Teams) | Mandatory for both (Every Team must compete in at least one Grand Prix, and every Grand Prix must have Teams participating). |
| Has a result in (Driver, Grand Prix) | Many-to-One (A Driver can have only one result per Grand Prix, but each Grand Prix has many Drivers with results) | Mandatory for Driver (Every Driver participating must have a result), Optional for Grand Prix (A Grand Prix can exist before results are recorded) |
| For (Ticket, Grand Prix) | One-to-One (Each Grand Prix has one recorded average ticket price, and each average ticket price corresponds to one Grand Prix) | Mandatory for Grand Prix (since every race has ticket prices), Optional for Ticket (if data is missing for some races) |
| Determines Standings for (Team, Championship) | One-to-One (Each Team has one final standing in one Championship) | Mandatory for Championship (every championship has a final team ranking), Optional for Team (if a team didn't participate) |
| Drives (Driver, Car) | One-to-One (Each Driver has one Car, and each Car is assigned to one Driver) | Mandatory for both (Every Driver must have a Car, and every Car must be assigned to a Driver) |

**ER Diagram:**

/

**RELATIONAL SCHEMA:**

**Championship** (Championship\_ID **PK**, Year)

**GrandPrix** (Race\_ID **PK**, GP\_Name, Date, Country, City, Circuit\_Name, Distance, Average\_Price\_USD, Championship\_ID **FK**)

**Team** (Team\_ID **PK**, Name, Country\_T, Principal)

**Driver** (Driver\_ID **PK**, Name, Nationality, DOB, Power\_Unit, Chassis, Model\_Name, Team\_ID **FK**)

**ParticipatesIn** (Team\_ID **PK, FK**, Race\_ID PK, **FK**)

**DriverReultsGP** (Driver\_ID **PK, FK,** Race\_ID **PK, FK**, Position, Points\_Earned, Fastest\_Lap)

**TeamStandings** (Team\_ID **PK**, **FK**, Championship\_ID **PK, FK**, Final\_Position, Total\_Points)

**Table Descriptions**

* + 1. **Championship Table:** Stores information about the F1 championships.
    2. **GrandPrix Table:** Stores details about individual Grand Prix races
    3. **Team Table:** Stores details about F1 teams.
    4. **Driver Table:** Stores details about drivers and their cars.
    5. **ParticipatesIn Table (removed)**: Stores information about which teams are participating in which Grand Prix races. This is a many-to-many relationship between the **Team** and **GrandPrix** tables.
    6. **DriverResultsGP Table:** Stores race results for each driver in a Grand Prix.
    7. **TeamStandings Table:** Stores the final standings of teams in a championship.

**Business Rules and Assumptions made**

(any logic that must be followed in database based on real world requirements)

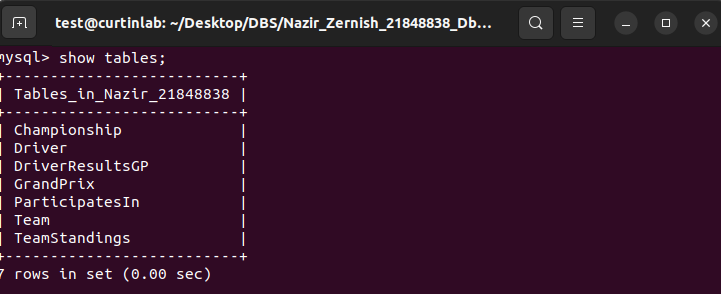
* Grand Prix must belong to a Championship there are no single races
* Drivers must belong to Team
* Each Grand Prix has an average ticket price which is within the Grand Prix table itself
* Drivers only drive a single car at a time
* Teams can participate in multiple races
* Drivers race results are recorded per Grand Prix, one result per driver per race
* Each team has a final standing position in the championship standings
* Used ONE DELETE CASCADE to ensure automatic deletion of records that are dependent when an FK entity is removed
* Not all race results were included for every Grand Prix due to the large number of driver results for each race. Only a subset of races and their corresponding driver results were added for the sake of simplicity and manageability in the initial implementation.

1. **Database implementation & Sampling Data**

**How I Implemented the database**

**Tables created**

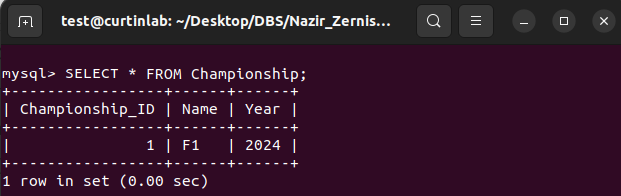
**Screenshots** of the tables created in MySQL



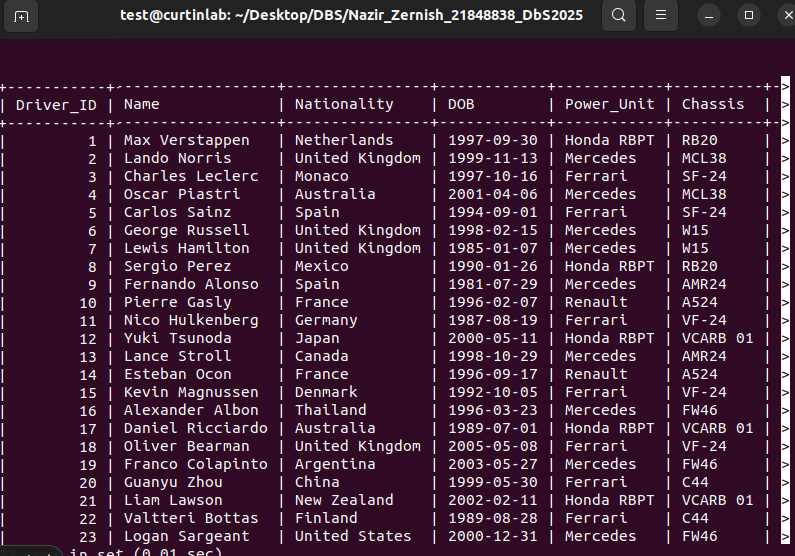
**Evidence of Implementation- Inserting data (more was added later on)**

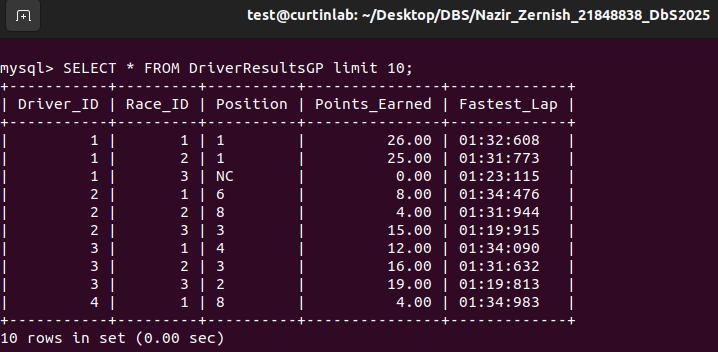
A sample SELECT \* FROM table output showing inserted data.

*Championship table*

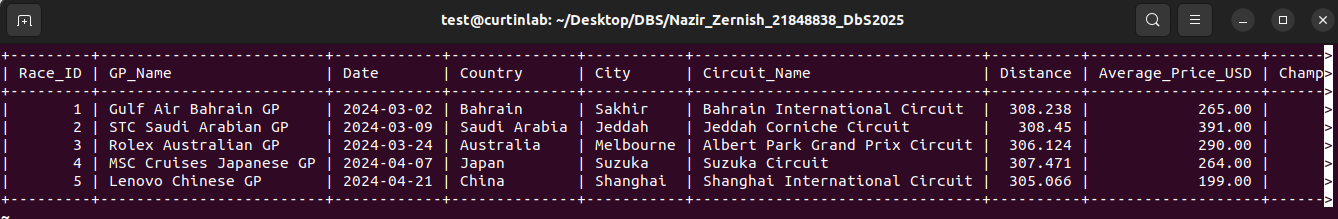


*Driver table*

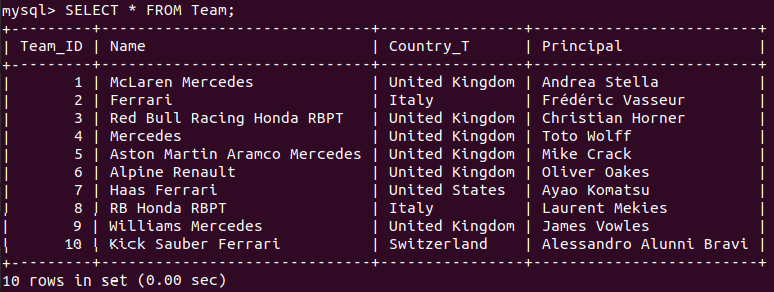


*DriverResultsGP Table*

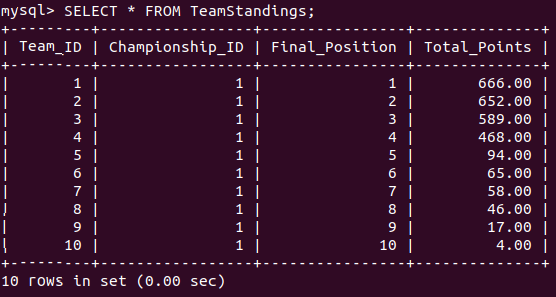
*GrandPrix Table*



*Team Table*



*TeamStandings Table*



**Description of sample data, data sources and how I inserted data into database**

The data was inserted into the database using SQL INSERT statements, with foreign key relationships ensuring consistency between tables. For example, the *TeamStandings* table was populated with final team standings based on actual 2024 results, that I got from the official F1 website.

As I was setting up my database, I realized that the *ParticipatesIn* table, which tracked team participation in each race, was redundant, as most teams participate in every race. I decided to simplify the schema by removing this table, reducing complexity and avoiding unnecessary data redundancy.

Moreover, instead of dynamically calculating the team standings through queries, I manually inserted the final team standings for the 2024 season, aligning with the assignment's focus on past data.

**Sources:**

I used two sources to populate the tables in my database. I chose the year 2024 F1 and inserted data relevant to that eyar into my database.

* **Github link (provided on blackboard):** <https://github.com/toUpperCase78/formula1-datasets/blob/master/Formula1_2024season_calendar.csv>
* **Official F1 site :** <https://www.formula1.com/en/results/2024/team>

1. **Using the Database**

**Design and implement queries**

**Level1:**

1. **Average ticket price for all Grand prix for Championship\_ID 1**

**Question**- What is the average ticket price across all Grand Prix events in the F1 Championship 2024 (first ID)

**Purpose:** this query can calculate the average ticket price for all Grand Prix races in the first championship (or can be changed to any other championship). Is useful for pricing strategies or understanding how the ticket costs vary. Can be useful to compare with other years, and helps plan future events.

1. **Find Races Held in a specific month (March)**

**Question-** Which Grand Prix races are held in the month of \_\_\_?

**Purpose:** Useful for even planning, promotions, and identifying seasonal race trends. What I wish to know by the query is simply the name of the GPs in the specified month.

1. **Find Drivers Whose Names Start with ‘L’ or who are Italian**

**Question-** Which drivers have names starting with ‘L’ or have Italian nationality?

**Purpose:** The query helps analyse drivers based on name patterns and nationality which may be useful for regional marketing, fan engagement, or just overall analysis.

1. **Convert Driver Names to Uppercase (With a Date condition)**

**Question-** show all the driver names in uppercase where the driver’s DOB is after January 1st, 1995

**Purpose:** The query is a demonstration of string manipulation using the upper function while filtering drivers born after 1995. This query could help analyse younger drivers competing in F1, useful query for reports and formatting. Helps highlight more recent drivers in a standardised format.

1. **Extracts the Year from Driver’s DOB**

**Question-** What is the birth year of each driver?

**Purpose:** Extracts the year of birth for each driver and orders the result by DOB, which can be helpful in age-based performance analysis, generational comparisons, and overall identifying youngest/oldest drivers in the sport. Also, easy chronological analysis

**Design and implement queries Level2:**

1. **Find the total points earned by each driver across all races**

**Question-** How many points has each driver accumulated across all Grand Prix races?

**Purpose:** The query helps analyse each drivers performance by calculating their total points. Its useful for determining overall rankings and evaluating their consistency across races

**Structure:** I used an INNER JOIN between the *DriverResultGP* and Driver to get each drivers name along with their total points. I used INNER JOIN since every result in *DriverResultsGP* must correspond to an existing driver. The SUM() function totals points per driver and I used GROUP BY *d.Name* ensures we calculate totals per driver. The ORDER BY *Total\_Points* DESC sorts drivers from highest to lowest points.

1. **Find the winning driver and fastest lap for each gran Prix**

**Question-** Who won each Grand Prix, and what was the fastest lap time in each case?

**Purpose:** The query will provide a clearer summary of each race, showing the winning driver and their fastest lap. Its useful for identifying performances that were better and race highlights.

**Structure:** I one INNER JOIN between *DriverResultsGP* and *Driver* to match each result to the driver, and another between *DriverResultsGP* and *GrandPrix* to retrieve the Grand Prix name. Since I only wanted the winners, I specified WHERE to be 1st position and added ORDER BY date so it sorted the races chronologically.

1. **Find the Fastest Lap for each grand Prix**

**Question-** What was the fastest lap time recorded in each Grand Prix?

**Purpose:** This query helps track race records by showing the fastest lap time per race. Its valuable information for performance analysis and identifying potential fastest drivers overall

**Structure:** Joins *DriverResultsGP* with *GrandPrix* to align the lap times with the races. To get the fastest lap per Grand Prix, I used MIN function on fastest lap which finds the lowest lap time for each race. GROUP BY *g.GP\_Name* ensures the aggregation happens per race and I ORDER BY *Fastest\_Lap\_Time* from fastest to slowest lap.

1. **Find drivers who have came in top 3 more than once**

**Question-** Which drivers have finished in the top 3 more than once?

**Purpose:** It highlights consistent top performers by counting how many times each driver finished in the top 3. Could help evaluate driver skill and competitiveness too.

**Structure:** I used an INNER JOIN between *DriverResultsGP* and *Driver* to retrieve driver names alongside their finishing positions. The where condition filters top 3 finishes, and I used COUNT to count how many times each driver finished in the top 3. HAVING COUNT(dr.Position) > 1 made sure it only show drivers who achieved this multiple times, while ORDER BY DESC ranks them accordingly.

1. **Clear view of team standings table**

**Question-** What are the final standings of each team in the F1 2024 Championship?

**Purpose:** The TeamStandings table mainly uses IDs since they are primary keys, making it difficult to interpret. This query provides a clearer output by displaying team names, the championship name, and total points.

**Structure:** join *TeamStandings* with *Team* and *Championship* to provide meaningful team standings instead of just IDs. INNER JOIN used since every team in standings table must have a corresponding entry in Team and Championship. The WHERE clause filters results to only show F1’s 2024 standings (in this case), while ORDER BY ts.Final\_Position ensures teams are displayed in ranking order.

**Design and implementation of advanced features**

I implemented:

* two stored procedures: with parameters, variables, IF-ELSE, and loops
* two triggers: BEFORE INSERT and AFTER UPDATE
* one complex view: involving many tables

**Stored Procedures**

**PROCEDURE 1: Calculate Average finishing position of a Driver**

**Question-** What is the average finishing position of a given driver across all races?

**Purpose:** this procedure when called helps analyse a drivers’ consistency. If their average is low, they are likely a strong competitor, gives an overview.

**Structure:** the procedure uses a JOIN to link *DriverResultsGP* with Driver to match the driver’s name. The AVG() function calculates the mean finishing position. Using IN parameters, I can input any driver name dynamically.

**PROCEDURE 2: Get races where a driver scored more than X points**

**Question:** In which races did a driver score more than a given number of points?

**Purpose:** Helps analyse high scoring performances, or just scores above specified point number across all races

**Structure:** This uses JOINs to link the *DriverResultsGP*, *Driver*, and *GrandPrix* tables. The WHERE clause filters the results based on the drivers name and the point threshold specified. The result is ordered by points in descending order to show the highest scoring races first.

**Triggers**

**TRIGGER 1: Automatically set a default fastest lap when inserting a new race result**

**Type of trigger:** BEFORE INSERT

**Purpose:** Ensures that if no fastest lap time is provided when inserting a race result, if defaults to 1:30:000 (1 min 30 sec).

**Structure:** The BEFORE INSERT trigger runs before inserting a new record into *DriverResultsGP*. If *Fastest\_Lap* is NULL, it automatically sets it to 01:30.000. This prevents NULL values and ensures data consistency.

**TRIGGER 2: Update the team points when a new driver result is inserted**

**Type of trigger:** AFTER INSERT

**Purpose:** Whenever a new result is added to the *DriverResultsGP* table, the total points of the team that the driver belongs to should be updated in the *TeamStandings* table. This ensures that the team’s total points correctly reflect the cumulative points earned by its drivers throughout the season.

**Structure:** The trigger ensures that the driver belongs to a team before updating team standings. It finds the *Team\_ID* of the driver who got a new result, adds the newly earned points to the team's existing total points in *TeamStandings*, and finally ensures that the update happens for the correct championship by matching *Championship\_ID* in *GrandPrix*

**Views**

**VIEW: Display detailed race results with Driver, team and Grand Prix Info**

**Question:** Can I see a detailed summary of race results, including driver, team, and race info?

**Purpose:** Makes it easier to analyse race results without joining multiple tables manually, simply select everything from the view.

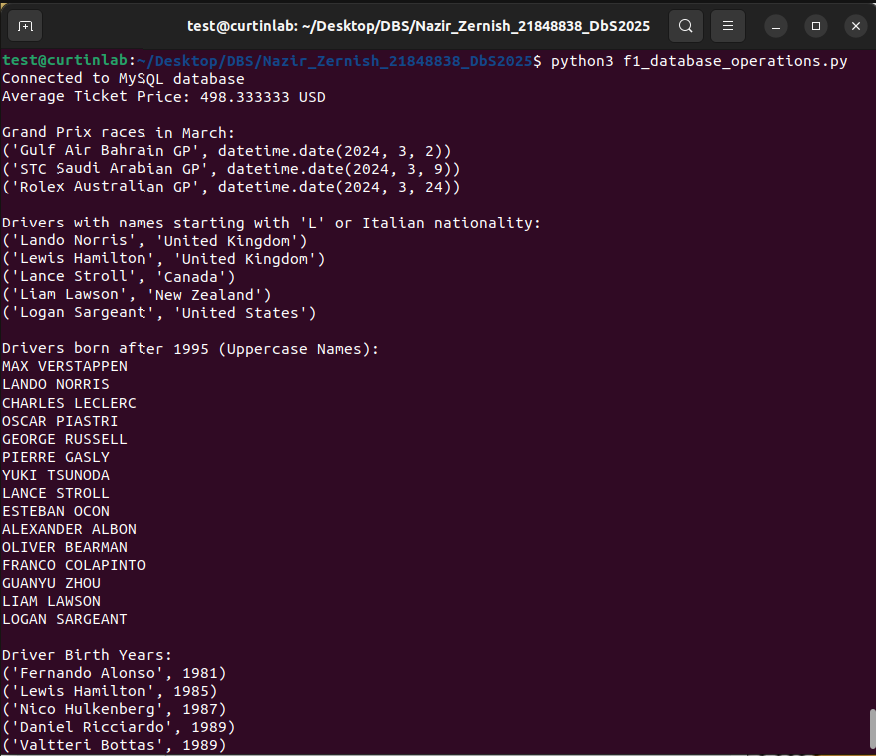
**Structure:** the view brings together DriverResultsGP, *Driver*, *Team*, and *GrandPrix* tables, providing a complete race summary. It orders the results by the race date (most recent first) and finishing position (best performances first).

**Database connectivity and Python implementation**

**Database connectivity description**

* For this part, I established a connection using mysql-connector-python between Python and MySQL. Ince I was connected I executed my predefined SQL queries by executing my python file. This ran SQL commands to retrieve and manipulate data
* I performed various operations, including selecting data, calculating aggregate values, and modifying records through insertion, updating, and deletion. Besides that my python script included my first five queries (the queries I wrote for level 1). These operations allowed me to interact with the database dynamically from a Python environment, ensuring data retrieval and management
* The connection was established by specifying the database credentials, including the hostname (localhost), user (f1\_admin), password, and the database name (Nazir\_21848838)
* I was able to automate database interactions and validate the functionality of the schema through real-time queries and updates.

**executing python code and output**

****

1. **Conclusion**

The database design process involved multiple adjustments and iterations to make sure it was efficient and consistent. Initially, certain redundant relationships, such as the ParticipatesIn table, were removed to simplify the schema. The Driver and Car entities were merged due to their one-to-one relationship, reducing unnecessary joins in queries.

During implementation, I added constraints such as ON DELETE CASCADE and CHECK conditions to maintain data integrity. Ensuring proper normalization (up to 3NF) prevented data anomalies, while foreign keys improved query performance.

I faced some challenges along the way. My second trigger didn't update the team standings points correctly upon insertion of a new result. This was because the trigger didn't verify if the driver had an assigned team or check the championship ID of the race, and I fixed it by adding NULL checks for the driver's team and properly linking races to their championships in the trigger logic. It retrieved the correct team id for the result inserted for the specific driver.

I also encountered issues running MySQL scripts because of authentication restrictions. Since MySQL on Ubuntu uses auth\_socket for root authentication (I used sudo before logging into MySQL), I was unable to connect using my Python script. To resolve this, I created a new MySQL user with the necessary privileges and updated my connection settings in Python. This ensured smooth execution of queries without requiring sudo access.

Overall, I learnt a lot about both Formula 1 and database implementation. Understanding the thought process and work that goes into it, and how teams, drivers and races are structured helped me make my database more realistic and efficiently. I’m also glad I gained hands on experience with enforcing constraints, writing more advanced SQL queries and troubleshooting all the issues I faced. Connecting MySQL to Python was a new challenge for me, but a very useful tool and helped me figure out how to manage and execute queries programmatically, an insight into real world database interaction